MTI #1757
Docket No. 18153.0045

VARIABLE CYCLE INTERRUPT DISABLING

FIELD OF THE INVENTION:

The present invention relates to systems and methods for processing interrupt and exceptions to instruction flow in processors and, more particularly, to systems and methods for providing variable cycle interrupt disabling.

BACKGROUND OF THE INVENTION:

Processors, including microprocessors, digital signal processors and microcontrollers, operate by running software programs that are embodied in one or more series of instructions stored in a memory. The processors run the software by fetching the instructions from the series of instructions, decoding the instructions and executing them. The instructions themselves control the order in which the processor fetches and executes the instructions. For example, the order for fetching and executing each instruction may be inherent in the order of the instructions within the series. Alternatively, instructions such as branch instructions, conditional branch instructions, subroutine calls and other flow control instructions may cause instructions to be fetched and executed out of the inherent order of the instruction series.

When a processor fetches and executes instructions in the inherent order of the instruction series, the processor may execute the instructions very efficiently without wasting processor cycles to determine, for example, where the next instruction is. When exceptions to normal instruction flow such as interrupts are processed, many processor cycles are taken away from the normal instruction flow to process an interrupt service routing (ISR) corresponding to the interrupt or exception.

In processor applications in which real-time performance of the processor is critical, there is a need to regulate when an interrupt is serviced in order to prevent impairing the real-time performance of the processor. The need may arise at only certain portions of a larger program, for example, when monitoring and processing operations are being performed. At these times, there is a need for a mechanism to prevent the servicing of an interrupt in order to devote processing power to processing the critical program portions.

SUMMARY OF THE INVENTION:

According to the present invention, a processor for processing a variable cycle interrupt disable instruction DISI X is provided. The instruction disables interrupt processing for a variable number of processor cycles corresponding to the value specified by the instruction operand X. The DISI X instruction may be strategically used by programmers to prevent interrupts from being taken during certain intervals within a program.

According to one embodiment of the invention, a method of processing an interrupt disable instruction includes fetching and executing an interrupt disable instruction that includes an operand specifying a number of cycles for disabling interrupt processing. The instruction operand may be the number or a pointer to the number. The method may further include changing the number based on processor cycles until the number reaches a predetermined value. The number may be changed by incrementing or decrementing with passing processor cycles. The method may include generating an interrupt disable signal during the changing of the number and generating an interrupt enable signal when the number reaches a predetermined value.

According to another embodiment of the invention, a processor includes an interrupt disable instruction processing feature. It includes a program memory, a register and an instruction fetch/decode unit. The program memory stores instructions, including an interrupt disable instruction DISI X, having an operand X specifying a number corresponding to an interrupt disable duration. The register stores the number and an instruction fetch/decode unit fetches and decodes instructions. When an instruction being processed is a DISI X instruction, the instruction fetch/decode unit decodes the DISI X instruction and disables the interrupt processing capability of the processor based on the number X.

BRIEF DESCRIPTION OF THE FIGURES:

The above described features and advantages of the present invention will be more fully appreciated with reference to the detailed description and appended figures in which:

Fig. 1 depicts a functional block diagram of an embodiment of a processor chip within which embodiments of the present invention may find application.

Fig. 2 depicts a functional block diagram of a data busing scheme for use in a processor, which has a microcontroller and a digital signal processing engine, within which embodiments of the present invention may find application.

Fig. 3 depicts a functional block diagram of a configuration for processing a variable cycle interrupt disable instruction according to an embodiment of the present invention.

Fig. 4 depicts a method of processing a variable cycle disable instruction according to an embodiment of the present invention.

DETAILED DESCRIPTION:

According to the present invention, a processor for processing a variable cycle interrupt disable instruction DISI X is provided. The instruction disables interrupt processing for a variable number of processor cycles corresponding to the value specified by the instruction operand X. The DISI X instruction may be strategically used by programmers to prevent interrupts from being taken during certain intervals within a program.

In order to describe embodiments of DISI X instruction processing, an overview of pertinent processor elements is first presented with reference to Figs. 1 and 2. The DISI X instruction functionality and processing is then described more particularly with reference to Figs. 3-4.

Overview of Processor Elements

Fig. 1 depicts a functional block diagram of an embodiment of a processor chip within which the present invention may find application. Referring to Fig. 1, a processor 100 is coupled to external devices/systems 140. The processor 100 may be any type of processor including, for example, a digital signal processor (DSP), a microprocessor, a microcontroller or combinations thereof. The external devices 140 may be any type of systems or devices including input/output devices such as keyboards, displays, speakers, microphones, memory, or other systems which may or may not include processors. Moreover, the processor 100 and the external devices 140 may together comprise a stand alone system.

The processor 100 includes a program memory 105, an instruction fetch/decode unit 110, instruction execution units 115, data memory and registers 120, peripherals 125, data I/O 130,

and a program counter and loop control unit 135. The bus 150, which may include one or more common buses, communicates data between the units as shown.

The program memory 105 stores software embodied in program instructions for execution by the processor 100. The program memory 105 may comprise any type of nonvolatile memory such as a read only memory (ROM), a programmable read only memory (PROM), an electrically programmable or an electrically programmable and erasable read only memory (EPROM or EEPROM) or flash memory. In addition, the program memory 105 may be supplemented with external nonvolatile memory 145 as shown to increase the complexity of software available to the processor 100. Alternatively, the program memory may be volatile memory which receives program instructions from, for example, an external non-volatile memory 145. When the program memory 105 is nonvolatile memory, the program memory may be programmed at the time of manufacturing the processor 100 or prior to or during implementation of the processor 100 within a system. In the latter scenario, the processor 100 may be programmed through a process called in-line serial programming.

The instruction fetch/decode unit 110 is coupled to the program memory 105, the instruction execution units 115 and the data memory 120. Coupled to the program memory 105 and the bus 150 is the program counter and loop control unit 135. The instruction fetch/decode unit 110 fetches the instructions from the program memory 105 specified by the address value contained in the program counter 135. The instruction fetch/decode unit 110 then decodes the fetched instructions and sends the decoded instructions to the appropriate execution unit 115. The instruction fetch/decode unit 110 may also send operand information including addresses of data to the data memory 120 and to functional elements that access the registers.

The program counter and loop control unit 135 includes a program counter register (not shown) which stores an address of the next instruction to be fetched. During normal instruction processing, the program counter register may be incremented to cause sequential instructions to be fetched. Alternatively, the program counter value may be altered by loading a new value into it via the bus 150. The new value may be derived based on decoding and executing a flow control instruction such as, for example, a branch instruction. In addition, the loop control portion of the program counter and loop control unit 135 may be used to provide repeat instruction processing and repeat loop control as further described below.

The instruction execution units 115 receive the decoded instructions from the instruction fetch/decode unit 110 and thereafter execute the decoded instructions. As part of this process, the execution units may retrieve one or two operands via the bus 150 and store the result into a register or memory location within the data memory 120. The execution units may include an arithmetic logic unit (ALU) such as those typically found in a microcontroller. The execution units may also include a digital signal processing engine, a floating point processor, an integer processor or any other convenient execution unit. A preferred embodiment of the execution units and their interaction with the bus 150, which may include one or more buses, is presented in more detail below with reference to Fig. 2.

The data memory and registers 120 are volatile memory and are used to store data used and generated by the execution units. The data memory 120 and program memory 105 are preferably separate memories for storing data and program instructions respectively. This format is a known generally as a Harvard architecture. It is noted, however, that according to the present invention, the architecture may be a Von-Neuman architecture or a modified Harvard architecture which permits the use of some program space for data space. A dotted line is

shown, for example, connecting the program memory 105 to the bus 150. This path may include logic for aligning data reads from program space such as, for example, during table reads from program space to data memory 120.

Referring again to Fig. 1, a plurality of peripherals 125 on the processor may be coupled to the bus 125. The peripherals may include, for example, analog to digital converters, timers, bus interfaces and protocols such as, for example, the controller area network (CAN) protocol or the Universal Serial Bus (USB) protocol and other peripherals. The peripherals exchange data over the bus 150 with the other units.

The data I/O unit 130 may include transceivers and other logic for interfacing with the external devices/systems 140. The data I/O unit 130 may further include functionality to permit in circuit serial programming of the Program memory through the data I/O unit 130.

Fig. 2 depicts a functional block diagram of a data busing scheme for use in a processor 100, such as that shown in Fig. 1, which has an integrated microcontroller arithmetic logic unit (ALU) 270 and a digital signal processing (DSP) engine 230. This configuration may be used to integrate DSP functionality to an existing microcontroller core. Referring to Fig. 2, the data memory 120 of Fig. 1 is implemented as two separate memories: an X-memory 210 and a Y-memory 220, each being respectively addressable by an X-address generator 250 and a Y-address generator 260. The X-address generator may also permit addressing the Y-memory space thus making the data space appear like a single contiguous memory space when addressed from the X address generator. The bus 150 may be implemented as two buses, one for each of the X and Y memory, to permit simultaneous fetching of data from the X and Y memories.

The W registers 240 are general purpose address and/or data registers. The DSP engine 230 is coupled to both the X and Y memory buses and to the W registers 240. The DSP engine

230 may simultaneously fetch data from each the X and Y memory, execute instructions which operate on the simultaneously fetched data and write the result to an accumulator (not shown) and write a prior result to X or Y memory or to the W registers 240 within a single processor cycle.

In one embodiment, the ALU 270 may be coupled only to the X memory bus and may only fetch data from the X bus. However, the X and Y memories 210 and 220 may be addressed as a single memory space by the X address generator in order to make the data memory segregation transparent to the ALU 270. The memory locations within the X and Y memories may be addressed by values stored in the W registers 240.

Any processor clocking scheme may be implemented for fetching and executing instructions. A specific example follows, however, to illustrate an embodiment of the present invention. Each instruction cycle is comprised of four Q clock cycles Q1 – Q4. The four phase Q cycles provide timing signals to coordinate the decode, read, process data and write data portions of each instruction cycle.

According to one embodiment of the processor 100, the processor 100 concurrently performs two operations – it fetches the next instruction and executes the present instruction. Accordingly, the two processes occur simultaneously. The following sequence of events may comprise, for example, the fetch instruction cycle:

Q1: Fetch Instruction
Q2: Fetch Instruction
Q3: Fetch Instruction
Q4: Latch Instruction into prefetch register, Increment PC

The following sequence of events may comprise, for example, the execute instruction cycle for a single operand instruction:

Q1: latch instruction into IR, decode and determine addresses of operand data
Q2: fetch operand
Q3: execute function specified by instruction and calculate destination address for data
Q4: write result to destination

The following sequence of events may comprise, for example, the execute instruction cycle for a dual operand instruction using a data pre-fetch mechanism. These instructions pre-fetch the dual operands simultaneously from the X and Y data memories and store them into registers specified in the instruction. They simultaneously allow instruction execution on the operands fetched during the previous cycle.

Q1: latch instruction into IR, decode and determine addresses of operand data
Q2: pre-fetch operands into specified registers, execute operation in instruction
Q3: execute operation in instruction, calculate destination address for data
Q4: complete execution, write result to destination

Variable Cycle Interrupt Disable Instruction and Processing

Fig. 3 depicts a functional block diagram of a configuration for processing a variable cycle interrupt disable instruction DISI X according to an embodiment of the present invention.

Referring to Fig. 3, an instruction fetch and decode unit 300 is coupled to a program memory 310. The program memory stores one or more programs with program instructions for execution. The programs may comprise different sections and subroutines, some of which have more stringent performance requirements than others. For example, there may be some subroutines or program sections that perform monitoring and processing functions, such as analog to digital signal conversion or other signal processing where the real-time performance of the processor must be very high. Other, subroutines or program sections may also have high real time performance requirements for other reasons. Programmers may also desire that certain

program sections, including interrupt service routines (ISRs), not be subjected to interruption during execution either because errors are likely or for other reasons.

To prevent interruption of a program, program section or portions thereof for a finite interval, a programmer may insert a DISI X instruction. The DISI X instruction causes the processor not to take an interrupt for a certain number of processor cycles defined by or corresponding to the operand X. The DISI X instruction may be placed anywhere in a program.

The instruction fetch and decode unit 300 decodes instructions, including the DISI X instruction, and dispatches the instructions and operands to the ALU 320, memory and registers 330 and the DSP engine 340.

When a DISI X instruction is received, the instruction fetch and decode unit 300 sends the operand X to a decrement register 350. The decrement register 350 may be associated with the interrupt logic 370. The decrement register receives the value of the operand X and is decremented automatically with each passing processor cycle. In one embodiment of the invention, the decrement register decrements the value in the register until it reaches a value of zero at which point the decrementing stops.

Variable cycle disable logic 360 is coupled to the decrement register 350 and to an interrupt logic and exception vector generation unit 370. The variable cycle disable logic receives the value in the decrement register. When the value is non-zero, the variable cycle disable logic generates an interrupt disable signal and transmits the signal to the logic 370. When the value is zero, the variable cycle disable logic 360 generates an interrupt enable signal and transmits the signal to the logic 370.

The interrupt logic and exception vector generation logic 370 receives interrupt request signals from various components of the processor and determines whether and when and to

service the interrupt requests based on the priority of the request. When the interrupt logic and exception vector generation logic 370 receives an interrupt disable signal from the variable cycle disable logic, it ceases to generate interrupts but the interrupts continue to be recognized and latched for later invocation.

A program counter 380 is coupled to the interrupt logic and exception vector generation unit 370. The program counter provides addresses to the program memory 310 which determine the next instruction fetched by the instruction fetch and decode unit 300. In the absence of an interrupt, the program counter 300 is incremented (unless a program instruction calls for a jump to a non-sequential instruction). When an interrupt is generated, the value in the program counter 380 is replaced with the address of instructions in the ISR corresponding to the interrupt generated.

When a DISI X instruction is processed, the value X is loaded into a decrement register, which is decremented in successive processor cycles until it reaches a zero value. While the value in the decrement register is not zero, the variable cycle disable logic generates an interrupt disable signal that disables the servicing of interrupts. Accordingly, many instructions in an instruction series after a DISI X instruction may be executed without interruption.

According to one embodiment of the invention, the decrement register is memory mapped. Accordingly, it may be read from and written to by other instructions. This feature may be used to enable interrupts again prior to the natural completion of a variable cycle interrupt disable. As an example, a variable cycle interrupt disable may be used during a subroutine. During or upon completion of the subroutine, an instruction may cause the decrement register to be written with a zero value to end the variable cycle interrupt on the next processor cycle. Alternatively, the subroutine may, upon completion or upon the occurrence of

another event, cause the decrement register to be written with a new value. The new value may be zero to terminate the interrupt disable or another value to perpetuate the variable cycle interrupt disable. The decrement register may also be read as any other memory mapped register.

Fig. 4 depicts a method of processing a variable cycle disable instruction according to an embodiment of the present invention. Referring to Fig. 4, in step 400 a variable cycle interrupt disable instruction DISI X is fetched form program memory. In step 410, the DISI X instruction is decoded. Then in step 420, the X operand of the DISI X instruction is loaded into a decrement register 350. In one embodiment, the value X is an immediate operand of the DISI X instruction and this value is stored directly into the decrement register 350. In other embodiments, X may be a pointer to a memory location or may identify a register which stores the value.

In step 425, the DISI X instruction disables interrupt processing. Then in step 430, the decrement register 350 decrements the value stored in it based on processor cycles. The decrement register may be set to decrement by one, two or any other value with each passing processor cycle. Moreover, the decrement register may decrement with each Q cycle or other signal on the processor. The decrement register may automatically stop decrementing when it reaches zero. Alternatively, a signal may be sent to the decrement register when the value reaches zero to stop the decrement register from decrementing.

In step 440, the processor determines whether the decrement register 350 has reached a predetermined value such as zero. If yes, then step 460 begins.

If not then step 430 begins again and the register value is decremented again. Steps 430 and 440 continue until in step 440 it is determined that the value in the decrement register has

reached zero. When this occurs, step 460 begins and the processor enables interrupt processing again.

While specific embodiments of the present invention have been illustrated and described, it will be understood by those having ordinary skill in the art that changes may be made to those embodiments without departing from the spirit and scope of the invention. For example, it will be understood that instead of a decrementing register 350, an incrementing register may be used. Alternatively, the value may be stored in a register that does not increment or decrement. Instead there may be a separate incrementing or decrementing register that produces values that are compared to the X operand of the DISI X instruction. In still other implementations, values other than zero may be chosen as a predetermined value to which to compare the incremented or decremented X value. All of these variations and others are within the scope of the invention.